Weapon Release Scheduling from Multiple-Bay Aircraft using Multi-Objective Evolutionary Algorithms

THESIS

Francis R Lyons, IV, 1st Lieutenant, USAF

AFIT/GCE/ENG/05-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/05-04

Weapon Release Scheduling from Multiple-Bay Aircraft using Multi-Objective Evolutionary Algorithms

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Francis R Lyons, IV, B.S.C.E.

1st Lieutenant, USAF

March 2005

AFIT/GCE/ENG/05-04

Weapon Release Scheduling from Multiple-Bay Aircraft using Multi-Objective Evolutionary Algorithms

Francis R Lyons, IV, B.S.C.E.
1st Lieutenant, USAF

Approved:

/signed/	7 Mar 2005
Dr. Gary B. Lamont (Chairman)	date
/signed/	7 Mar 2005
Dr. Gilbert Peterson (Member)	date
/signed/	7 Mar 2005
Dr. Meir Pachter (Member)	date

AFIT/GCE/ENG/05-04

Abstract

The United States Air Force has put an increased emphasis on the timely delivery of precision weapons. Part of this effort has been to use multiple bay aircraft such the B-1B Lancer and B-52 Stratofortress to provide Close Air Support and responsive strikes using 1760 weapons. In order to provide greater flexibility, the aircraft are carrying heterogeneous payloads which can require deconfliction in order to drop multiple different types of weapons. Current methods of deconfliction and weapon selection are highly crew dependant and work intensive.

This research effort investigates the optimization of an algorithm for weapon release which allows the aircraft to perform deconfliction automatically. This reduces crew load and response time in order to deal with time-sensitive targets. The overall problem maps to the Job-Shop Scheduling problem. Optimization of the algorithm is done through the General Multiobjective Parallel Genetic Algorithm (GENMOP).

We examine the results from pedagogical experiments and real-world test scenarios in the light of improving decision making. The results are encouraging in that the program proves capable of finding acceptable release schedules, however the solution space is such that applying the program to real world situations is unnecessary. We present visualizations of the schedules which demonstrate these conclusions.

Acknowledgements

I owe an incredible debt of gratitude to my wife, Audra Lyons. Without her love and support, this effort would not have been successful. I also wish to acknowledge the support of my family, who pretended to be interested in my topic for far longer than they had to.

Francis R Lyons, IV

Table of Contents

List of Figures

Weapon Release Scheduling from Multiple-Bay Aircraft using Multi-Objective Evolutionary Algorithms

I. Introduction

The changing nature of warfare has brought about a revolution in the accuracy and precision of weapons used by the United States Air Force. Where more than 600 weapons were required in 1944 and 175 weapons were required during the Vietnam conflict, current operations require only a single weapon [33]. With many different types of weapons available, the capability of carrying multiple weapons to provide the single weapon necessary is of great utility. The ability to drop multiple types of weapons during a single pass is currently restricted to multiple-bay aircraft such as the B-1B Lancer, B-52H Stratofortress, and B-2 Spirit bombers. This particular investigation explores the optimization of the release orders of various weapons in order to minimize threat exposure time as well as bay door open time.

The enhancement of weapon system effectiveness has been in large part due to the use of so-called "smart weapons". These smart weapons are those weapons which are able to perform some type of target tracking or other terminal guidance through the use of internal systems or external guides. For the purposes of this investigation, smart weapons refer to those weapon systems which utilize a MIL-STD-1760 connection for data transfer and weapon status monitoring [4]. Currently, almost all aircraft currently in Air Force inventory may make use of 1760 weapons, however we are constraining our investigation to multiple bay aircraft.

One of the changes in the use of weapons has been in the use of different weapons for different types of targets. Where previously only certain types of weapons were available as precision-guided munitions, now there are multiple types

Weapon	Aircraft		
	B-1B	B-52H	B-2
AGM-154 Joint Standoff Weapon (JSOW)	X	X	X
GBU-31 Joint Direct Attack Munition (JDAM)	X	X	X
Wind Corrected Munition Dispenser (WCMD)	X	X	
AGM-158 Joint Air to Surface Standoff Missile (JASSM)	X	X	X

Table 1.1: 1760 Weapons

which can be employed. Some of these weapons and the aircraft under consideration are listed in Table 1.1. All of these weapons have different use requirements and characteristics in terms of altitude, airspeed, door position, and positioning depending on the platform.

Previously, multiple bay aircraft could only carry and deliver a single type of weapon system (e.g. a homogeneous environment). There are two possible ways in which a system could be heterogeneous. The first is referred to as "bay-pure," in which each bay on the aircraft has the same type of weapon, but the bays have different types. A more difficult problem is one in which weapons are mixed not only across the bays but within the bays as well. The system analyzed in this research is that of a heterogeneous environment in which different types of weapons are carried within as well as across the bays.

1.1 Sponsors

This research effort is sponsored by the Air Vehicles Directorate (VA) and the Munitions Directorate (MN), Air Force Research Laboratory (AFRL), Eglin Air Force Base, Florida. The mission of AFRL/MN is to "develop, integrate, and transition science and technology for air-launched munitions for defeating ground fixed, mobile/relocatable, air, and space targets to assure the pre-eminence of U.S. air and space forces." [2] The research contained herein supports the stated mission through the development and analysis of an algorithm which allows for shorter response times in order to defeat enemy targets. Specific points of contact include Mr. Lloyd Reshard (AFRL/MN) and Dr. Gary Lamont (AFIT/ENG).

1.2 Research Goals and Objectives

The goal of this research is develop an algorithm that efficiently and effectively schedules the release order of different weapons based upon changing objectives. Based upon this formulation of the problem, it can be decomposed into several primary objectives:

1. Develop a model that encapsulates the problem domain.
2. Develop an effective algorithm which solves the scheduling problem.
3. Evaluate the algorithm for efficiency.
4. Provide the algorithm in portable form for use in multiple environments.

1.2.1 Objective 1: Model Development.

The goal of this program cannot be accomplished without understanding the problem domain in which we are operating. While the English language is sufficient to communicate the problem domain, it is necessary to quantify it symbolically, using mathematics. The analysis of the problem domain provides the necessary information to create the mathematical formulation.

The mathematical formulation during the initial phases is a general, high-level abstraction of the problem domain. As further studies are conducted, the formulas and descriptions become more precise, providing a more accurate model to be used. The definition of the model includes defining objective functions and constraints related to the problem domain.

1.2.2 Objective 2: Algorithm Development.

In order to complete the research, an algorithm is developed which is capable of providing solutions to the model of our problem (e.g. a schedule for the proper sequence of weapons to be delivered). A key portion of this objective is that the algorithm must effectively solve the scheduling problem, which implies that the developed algorithm must find "optimal" solutions either more often or more rapidly than prior existing algorithms.

The plan develops an algorithm and evaluates its effectiveness. Using the mathematical formulation from our first objective, we utilize a deterministic algorithm to solve low-dimensional problems such as the bay-pure model. Using the previous results, we develop a stochastic algorithm to solve problems of higher dimensionality. The results from the stochastic method are compared with the solutions for the pedagogical problems which are used as test material. When the stochastic method is capable of generating optimal solutions, the results are compared with other results from the literature.

1.2.3 Objective 3: Algorithm Optimization. With the successful development of an effective generic algorithm capable of solving the problem, the next step is tuning the algorithm to address the results of the refined model. This step improves the efficiency of the algorithm, providing increased performance during operational use.

The initial runs of the algorithm are used as a baseline for all future implementations of the algorithm. One issue in this research is that the final objective is a generic algorithm which is capable of be implemented with relative ease on heterogeneous platforms of varying processor speed and memory capacity. Therefore, instead of only looking for any bottlenecks which exist in the implementation of the algorithm, research is also focused on making the algorithm itself more streamlined. This includes efforts such as exploring the parallelization of the algorithm.

1.2.4 Objective 4: Algorithm Portability. With development and optimization of the algorithm completed, the goal is to provide the algorithm in a portable form to the user. This requires that the algorithm be repackaged from the developmental work into a form, such as an executable, that can be used without significant knowledge of the processes that it uses to solve the problem. With development proceeding in a Linux-based, parallel environment, it also needs to be usable in a single-processor, non-Linux environment to provide the greatest ease of use possible.

1.3 Approach

Our approach to this problem is straightforward. The research looks at and discusses the problem domain in detail. We outline the basic approaches to a more general problem which is then mapped to our problem domain, and point out some of the previously used solutions. A prototypical solution is chosen, and detailed design is done using our knowledge of the problem domain. Tests are run using pedagogical examples as well as test sets from actual multi-weapon aircraft flights. The results of the tests are analyzed, and conclusions are presented based on this analysis.

1.4 Thesis Overview

Chapter II gives background information and formulations of the overall problem and the Job-Shop Scheduling problem. It outlines the various approaches to the Job-Shop Scheduling problem as well as outlining our method for solving the problem. Chapter III describes the low-level, detailed analysis of the algorithm domain and the design of algorithms for the problem. Chapter IV describes implementation of the design and the design of experiments to validate the algorithm. Chapter V discusses the results of the experimental work and analyzes the data in terms of our objectives. Chapter VI presents the conclusions from the research with an eye to future work.

II. Background: Job Shop Scheduling Problem & Genetic Algorithms

The problem of performing optimization has been addressed in a variety of environments. In this chapter, we examine some of the fundamental background and approaches which have been used to resolve problems of the same type.

Our generic problem with constraints is mapped to a general problem set, that of the Job-Shop scheduling problem. The Job-Shop problem is described and potential approaches are outlined. This lays the foundation for future discussion of the mathematical model.

2.1 Problem Description

The generic problem we are trying to solve is as follows. Air Force multiple bay aircraft have traditionally used a homogeneous loadout of weapons[1]. Reasons for this included needing multiple weapons of the same type to be dispatched at a single target in order to have an acceptable probability of a kill. In the face of increasing accuracy, it became less necessary for multiple weapons to be delivered at the same target; however, limitations in the avionics hardware and software continued to constrain weapon loadout.

With recent advances in technology, this constraint has been removed. It is increasingly desirable that multiple-bay aircraft, which are referred to as "bomb trucks," be capable of carrying multiple types of weapons, in particular smart weapons. An example can be seen in Figure 2.1 in which a B-1B Lancer drops MK82 bombs from all three bays. This is to provide local commanders with greater flexibility in calling for support and allows the aircraft to support a much wider range of missions. In fact, the former limitation of bombers to strategic attack has been removed, allowing the aircraft to function even in close-air-support roles [44]. The inherent

[1]A homogeneous weapon load is one in which the aircraft is carrying the same type of weapon in of the bays of the aircraft.

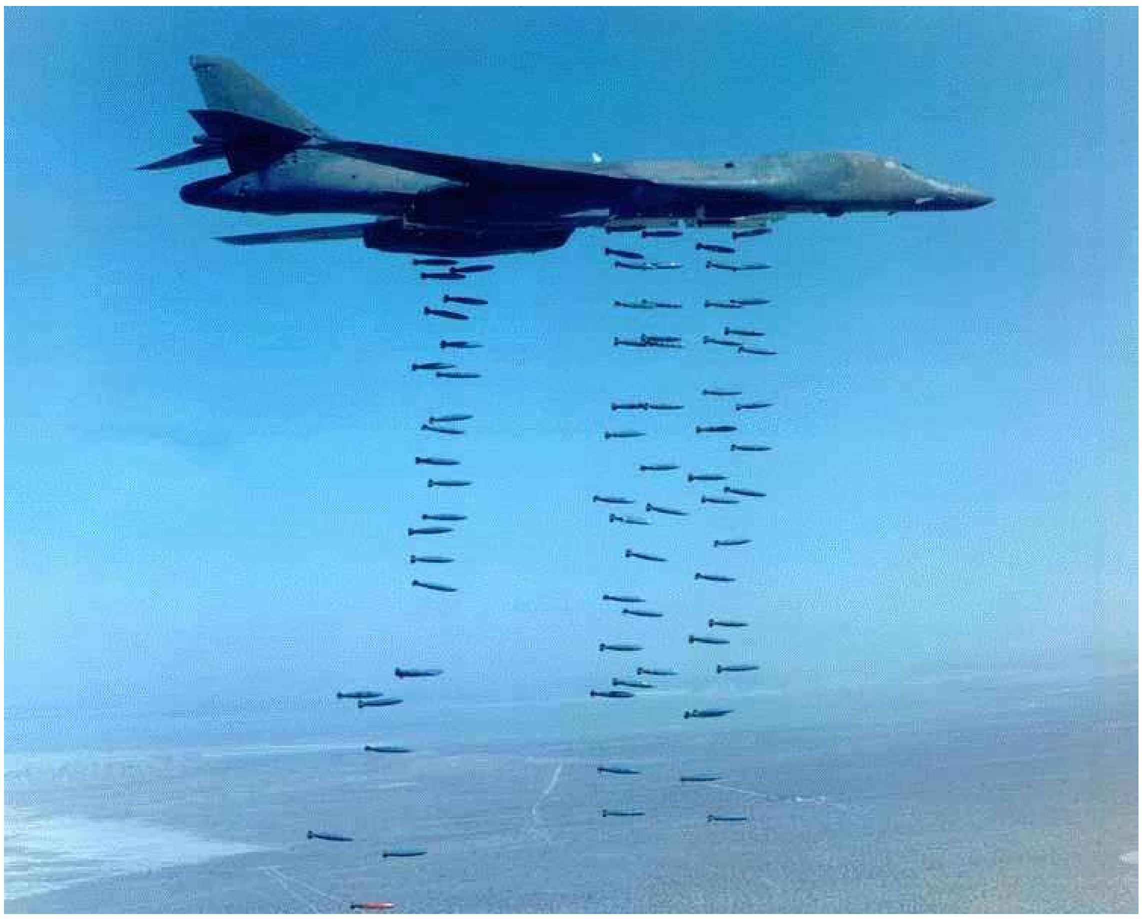

Figure 2.1: B-1B Lancer showing the three bays

inability to plan out loitering missions, where the aircraft is waiting for targets to be relayed to them, means that weapon assignment and scheduling needs to be performed dynamically.

The problem that we are attempting to solve is that of the scheduling the release of multiple types of weapons concurrently. The overall goal is to minimize the amount of time that the bay doors are open, since the buffeting of the doors caused by the speed is a critical limiting factor in the service life of the bay doors. Additionally, the problem needs to be able to maximize the airspeed of the aircraft to provide for low threat exposure time as well either maximizing or minimizing the altitude based on the mission profile.

As outlined in Chapter I, our first objective in this research is to define the problem domain in which we are operating. This chapter is primarily concerned with providing the background on our problem domain mapping and potential solutions. From the preceding description of the generic problem, we can map to a general domain where there are known approaches to finding a satisfactory solution. In this area, we map our problem to the general problem domain known as the Job-Shop scheduling problem.

2.2 Job-Shop Scheduling Problem

The Job-Shop scheduling problem is part of a class of computational problems known as "NP-complete" problems. This large set of problems is defined by either all or none of them having polynomial time solutions, with strong evidence that there is no polynomial time solution [70]. The NP-complete set of problems is "hard" in that the problems are ones "for which we cannot *guarantee* to find the best solution in an acceptable amount of time." [28] Having an NP-complete problem entails using a heuristic to get an approximation of the optimal solution. From the discussion in [70], we know that not only is the scheduling problem NP-complete, but also that it remains NP-complete for $\prec = \emptyset$ and $\Re = 2$. [70], Section 4.7, details the proof.

In this section, we discuss the formulation of the Job-Shop scheduling problem, its application to the problem at hand, and some algorithms which have been used to solve it.

2.2.1 Formulation.

The Job-Shop scheduling problem is summarized scheduling a number of jobs on a set of machines such that the makespan, the time to complete the last job, is minimized. The general task system in mathematical form is given by [27], by the following tuple:

$$(\Im, \prec, [\tau_{ij}], \{\Re_j\}, \{w_i\})$$

$\Im$ is the set of tasks to be completed with $\prec$ defining the partial order in which those tasks must be completed. $[\tau_{ij}]$ is an $m \times n$ matrix of the execution times where i denotes the job and j denotes the processor. When $\tau_{ij} = \infty$, it signifies that job cannot be completed on that processor. $\Re_j$ is the set of the amount of resources that a job requires. The canonical Job-Shop scheduling problem defines $\Re_j$ in terms of the number of processors or machines that are required, in the mapping to our general problem, $\Re_j$ is defined in terms of the number of door movements which must be made. Finally, w_i is the cost rate or deferral cost for completing $\Im_i$ at some time t.

There are two primary methods of describing scheduling algorithms, based upon the ability of the schedule to be manipulated. The first is that of list scheduling. This type of scheduling assumes that $\Im$ is constructed as an ordered list. In combination with $\prec$, this means that the priority list serves as the basis for the scheduling, with no out of order execution allowed. Scheduling is therefore performed by repeated scans of the list whenever a processor becomes free for assignment [27]. This type of scheduling algorithm is inappropriate for our problem since we specifically want to allow for reordering of delivery order to minimize bay door open time.

Since our jobs in the problem must be allowed to complete before another job is scheduled, we use nonpreemptive scheduling. The alternative type of scheduling allowed for preemption, that is, jobs could be interrupted based on the fact that at some point it would receive all of the necessary processor time [27]. We cannot use this type of scheduling since the process of dropping a weapon has very few points during which it can be delayed in order to allow another weapon to proceed.

2.2.2 Schedule Metrics. Once a schedule is generated, we must be able to compare it to other schedules. Each schedule has a flow time, denoted $f(S)$ where S is the schedule. Since this is a maximum for a given schedule, we want to find the minimum member of a set of maximums, a classic optimization problem. This is represented in Equation 2.1, adapted from [27].

$$\omega(S) = \min(\max_{1 \leq i \leq n}\{f_i(S)\}) \tag{2.1}$$

This problem is made more difficult by not allowing for preemption. Since we are using identical processors in this case, the bays, the minimum length schedule is optimal when $\prec$ is empty [27]. While our problem often allows for this case, the assumption cannot be made that $\prec$ is empty by the discussion in Section 3.2.1.

2.2.3 Algorithms. As algorithms can be divided into deterministic and stochastic classes, we examine both for algorithms capable of solving the scheduling problem. Although there is some discussion in [26, 64] about algorithms which are used to give approximations for the scheduling problem, the nonpreemptive discussion is limited to those task sets which can be formulated as directed acyclic graphs (DAG) or where the number of processors is greater than two.

The deterministic approach to scheduling has been summarized by Kwok and Ahmad [48]. The most common methods are list scheduling techniques in which the $\prec$ operator is used to determine the schedule. An example of this class of algorithms is the Insertion Scheduling Heuristic (ISH) [47] which operates on DAGs with communication delays by inserting any ready tasks into slots which exist due to communication delays. The Duplication Scheduling Heuristic (DSH) [47] is an extension of the ISH which uses task duplication to reduce the start time of tasks within a schedule [77]. Other deterministic methods to solve the scheduling problem include Tabu Search [16, 62] and the shifting bottleneck procedure [9, 15, 16].

Our area of interest is in using genetic algorithms (GAs) to provide an acceptable solution to the task scheduling problem. We refer to an acceptable solution rather than to solving the problem since solving implies that a definitive, optimal solution is found. We anticipate finding several solutions and choosing between them. As noted in [77], there have been a multitude of ways in which GAs have been applied, generally falling into two main approaches. The first approach uses GAs as helper functions in combination with other list scheduling techniques to produce schedules. The other method is to utilize the GA to "evolve the actual assignment and order of tasks into processors." [77] Here, we use the second approach to generate a schedule based upon a list of weapons to be dropped. We choose this algorithm based upon the results available in [72,73] that demonstrate the capability of genetic algorithms in solving dynamic and static scheduling problems.

2.3 Evolutionary Computation Domain

The field of evolutionary computation has demonstrated satisfactory results in solving hard problems such as the Job-Shop scheduling problem. Evolutionary computation's optimization process allows us to discover "highly precise functional solutions" to a particular problem [32], which is the objective of this research. For this reason, we use evolutionary computation to solve our general problem. The definition of evolutionary computation is that it uses the methods of natural evolution to solve problems. Populations of solutions are evolved so that at some point, the best solution found is considered as the individual with the highest fitness. [6]

At the most general level, search algorithms are algorithms which "search" either the problem space or the solution space of a problem to find the optimal answer. At the highest level, the interest is in exploring the search space. This exploration occurs deterministically or stochastically, providing the foundation for the categorization of search algorithms. Deterministic algorithms are those which, given the same starting point, find an optimal or satisfactory solution every time. Stochastic algorithms are random searches across the solution space which may not find a satisfactory solution each time. Additionally, stochastic searches allow us to exploit any partial solutions which we have found, since those partial solutions could potentially be part of the final solution. Evolutionary computation (EC) is an inherently stochastic area of study, with the randomness of the search discussed in Section 2.3.6.

2.3.1 Evolutionary Algorithms. According to [12], the scheduling problem is just one of several combinatorial problem which can benefit from the use of evolutionary algorithms. Other areas include:

- Routing
- Packing
- Design
- Simulation
- Control
- Classification

With all of these areas benefitting from evolutionary computation, it is important to understand some of the concepts behind the term "evolutionary algorithms."

The natural process of evolution is itself an optimization process [32]. While not always generating perfect solutions, evolution is quite capable of developing good functional solutions to a particular problem environment. When faced with problems whose chaotic nature did not lead themselves to being solved deterministically, computer scientists and engineers began applying the essential principles of natural evolution, that is, growth and adaptation over time, to the computing domain. Please see [6] for a more in depth discussion of this topic. We are primarily concerned with the basic operators of evolutionary computation, which are best understood from the biological context in which they developed.

The EC efforts use many of the same terms as the biological descriptors of evolution. As in biology, the fundamental unit of information is the chromosome. As outlined in [42], the discovery of chromosomes as the "carriers of genetic information" allowed biologists to begin exploration into how that information was transmitted and passed on to future generations. EC research uses chromosomes in much the same way, with the particular problem domain affecting how the chromosome is defined and described. Within an experiment's population, each individual is a chromosome in biological terms.

Biologists have long known that an individual is best described in terms of its phenotype and genotype. [11] In EC, as in biology, the term phenotype refers to the appearance of an organism or individual. In computation, this means that, for example, an individual has a value of 0.45 in the range of 0.00 to 1.00. [24] The genotype describes the DNA of an individual, and so is the primary mechanism for "variance within a population because genetic changes caused by mutation and recombination are passed with the genome." [11] In the job-shop problem, the genotype is the set of jobs that form a given schedule. The phenotype is how those jobs are described in terms of the specific problem, such as a problem with five jobs to be scheduled being constrained to the range from one to five inclusive.

Evolutionary algorithms are so named because they use the operators of biological evolution acting upon possible solutions to explore the problem domain and exploit any "better" solutions. [6, 11] The basic algorithms in evolutionary computation can be separated into two types, based upon their dependence upon the different fundamental operators of evolution. The first operator is that of mutation, and evolutionary strategies and evolutionary programming rely primarily upon this operator to provide the development of a solution. The second category is that of genetic algorithms and genetic programming, which are focused upon the crossover (or recombination) aspect of evolution. Many evolutionary computation efforts in fact use both crossover and mutation to achieve the best results in exploration and exploitation of the solution domain.

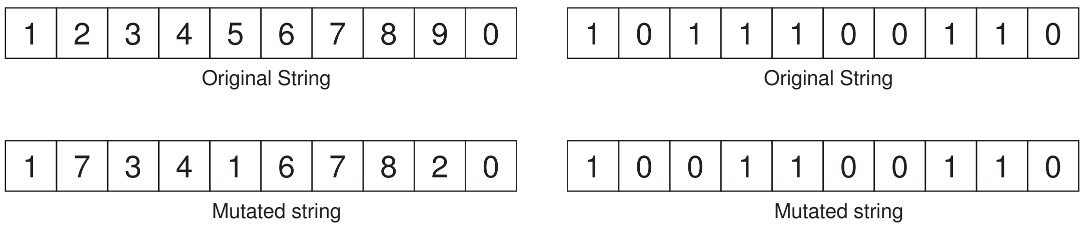

Figure 2.2: Example of Mutation

2.3.2 Mutation. In biology, mutation performs a vital operation in that it acts upon a single chromosome or individual. The genotype is the focus of the mutation operation. A biological mutation changes one of the pieces of genetic information into another type of information. For example, the DNA sequence AGTTA may become AGGTA. The computational version of mutation does the same to the genotype of one of the individuals. Examples of this can be seen in Figure 2.2. Much like in biology, EC uses mutation sparingly for the most part, with genetic algorithms such as we are dealing with using a low percentage ($\leq 1\%$) of mutation. An evolutionary strategy, on the other hand, uses mutation as its primary operator. For continued discussion of evolutionary strategies, please refer to [63].

As discussed in [8], there are many ways in which mutation can be applied to the many formulations of problems in evolutionary computation. The underlying biological premise is simple. As in nature, mutation is a change in the genotype domain of an individual within a population. Possible ways to instantiate this type of operator are bit flipping, position swapping, and permutations [8].

Mutation allows for increased exploration of a search space, since it moves an individual from a given location within the solution space to another. How the chromosome is defined and the specific mutation operator operates determines how far a potential mutation would move the individual.

One problem that can arise in using the mutation operator is that duplication of a phenotypic characteristic can occur. In some problem domains, this is not an

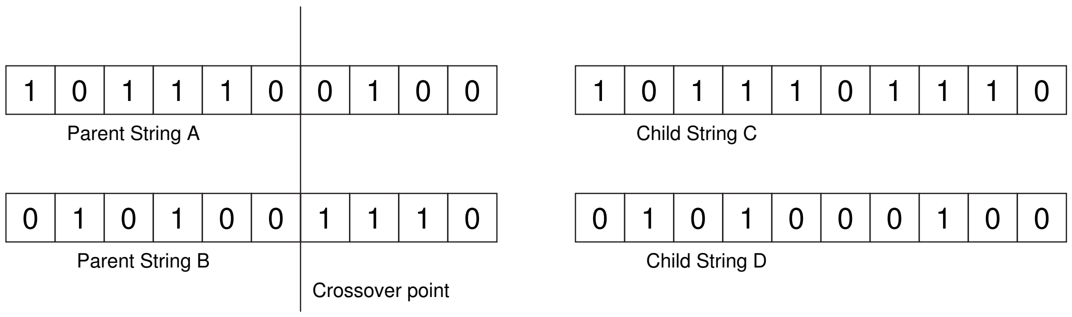

Figure 2.3: Example of crossover

issue. However, in the scheduling problem, this cannot be permitted. It is intrinsic to the problem that we do not want to have the same job scheduled multiple times. In this case, instead of changing a single location, the corresponding location with the same value must be mutated to have the value in the previous position.

2.3.3 Crossover. Also known as recombination, crossover is the practice of taking individuals from a "parent" population and swapping pieces between the individuals to create children for the next population. In biology, the chromosomes undergoing recombination are aligned, a breakage occurs at the same location in both, and the "homologous chromosome fragments are exchanged." [18] This recombination of material provides variability in the population.

Our description of the basic crossover operation is taken from [18], and was introduced in [39]. It is composed of three steps. A simple, one-point crossover is illustrated in Figure 2.3

1. Two individuals are chosen at random from the population of potential 'parent' strings.
2. One or more points in the chromosomes are chosen as crossover points, creating substrings to exchange.
3. The substrings are exchanged and combined, creating two 'offspring.'

It is possible to extend the general principle of two parent strings to allow for multiple parent crossover. [18] highlights some efforts in this area. Additionally, the number of offspring is not constrained to simply two. In [39], one of the children in thrown away, retaining a single child, while there are algorithms that produce many more offspring and retain each of them.

This operation provides the primary means of exploitation in evolutionary computation. An individual is staying within an area of the search space, but it is moving within that area. This allows a potential optimum location in the search landscape to be exploited. A critical assumption is that the individuals being used in the crossover information have higher quality building blocks then the rest of the individuals [11], taking advantage of Holland's schema theorem. [39]

2.3.4 Constraint Handling Techniques. Part of the definition of any problem are the constraints inherent to that problem. In our problem domain, the constraints determine whether a solution is infeasible or feasible. Since feasible individuals by definition are legitimate solutions, the question then is one of how to handle the infeasible individuals. This area of evolutionary computation is called constraint handling.

Constraint handling focuses on dealing with those infeasible individuals produced by the stochastic operation of an evolutionary algorithm. Potential ways of addressing this are penalizing infeasible individuals, changing the topology of the search space, repairing infeasible individuals, or only have feasible individuals in the initial population [56]. The first method is to use a penalty function to make it more difficult for infeasible individuals to remain in the population [66]. The penalty function can actually work on either feasible or infeasible individuals, depending on the problem topology. Additionally, the function can be static, with predefined penalties for distance from a desired area, or dynamic, with penalties changing as the solution space is explored. The dynamic version allows for infeasible individuals early in the exploration, but weeds them out at the end [66].

By changing the topology of the search space, we greatly reduce the number of infeasible individuals which can be generated in the population. This transformation decodes the chromosomes in order to know how to build a feasible member [55]. Basically, it is a mapping from the entire search space into a feasible subspace [55]. Another means of dealing with undesirable members of the population is to repair those individuals. What this means is that we take the portions of an individual which violate some constraint, and change it so it no longer violates that constraint [58]. The repair is usually done by finding the closest feasible point to that member, and moving it to that point. Repair algorithms can be computationally expensive to perform since there are additional calculations which must be performed in order to move the individual.

There are many other algorithms for dealing with infeasible individuals. [25,54, 57] all deal with the aforementioned methods as well as others which are of interest.

2.3.5 GENMOP. The GENeral Multi-Objective Program (GENMOP) was developed at the Air Force Institute of Technology (AFIT) to provide an implementation of a genetic algorithm. [46] Subsequent efforts at AFIT expanded GENMOP to be run in parallel on a cluster of computers. GENMOP is designed to be as modular as possible, with a user defining the chromosome representations, fitness function, and genetic operators to use. The basic algorithm, in Bäck's notation, is given in Algorithm 1. Of particular interest is the crossover and mutation operations of GENMOP. The algorithm chooses from four possible crossover functions and two possible mutation functions based upon a normalized distribution for each.

Algorithm 1 GENMOP in Bäck's Notation

```
1: $t := 0;$
2: $P(0) := \{a_i(0), ..., a_\mu(0)\} \in I^\mu$
3: evaluate $P(0) : \{\Phi(a_i(0)), ..., \Phi(a_i(0))\}$ where $\Phi(a_k(0))$ is user defined.
4: for $i = 1$ to Max_num_of_generations do
5:     $P_i := null;$
6:     $P_m := null;$
7:     $P_m := P_{i-1}$
8:     $P_i := P_m$
9:     $P_m :=$ crossover($P_i$)
10:    $P_i :=$ mutate($P_i$)
11:    $P_i := P_i + P_{i-1} + P_m$
12:    evaluate($P_i$)
13:    normalize($P_i$)
14: end for
```

The symbols in GENMOP are described thusly. P is a population, with $P(0)$ being the population at time zero. An individual (a) is within the range described by I^μ, with μ being the number of values in the range. Φ is a fitness function which is applied to the each of the individuals in the population as an initial evaluation step. P_m is the mating pool for the algorithm, and crossover is performed upon this population while P_i takes the previous generation and performs mutation on its members. The old population, the children of crossover, and the mutated population are combined. The combined population is ranked and normalized to maximum population size. The algorithm continues until the maximum number of generations is reached.

2.3.6 Random Number Generators. As we discussed in Section 2.3.5, the use of random number generators (RNGs) in genetic algorithms is important, if not critical. For this reason, we must be satisfied that the random number generator(s) in use provide sufficient "randomness" to satisfy the stochastic nature of GAs. For any RNG, the important factors in its evaluation are its distribution and its periodicity. For an extended discussion of these factors, see [59, 74].

The RNG in use in GENMOP is found in `RNG.cc`. It was developed as part of the GNU Library in 1989 by the Free Software Foundation. This implementation has been used in many other applications, and we therefore make the assumption that it has been validated to provide sufficient randomness.

2.4 Summary

In this chapter, we discuss the outlines of the general problem domain, with application to our particular problem. The fundamental concepts and operations for EC are introduced and explained. The problem is described literally and mathematically, with a presentation of the initial mappings into the EC domain. We next present the development of our problem domain in the context of what is outlined in this chapter.

III. Problem Domain Development

The focus of this chapter is provide a design for a solution of the scheduling problem described in II. Additional background is provided for the specific problem of weapon scheduling from multiple bay aircraft. Constraints present in the problem are specified, and aircraft specific factors are identified. Fitness functions for evaluating the population of our genetic algorithm are designed.

3.1 Design goals

In Section 2.1, the overall problem that we address is described. The focus of this chapter is on utilizing the techniques and approaches outlined in Chapter II to provide a solution for our general problem.

3.2 Problem Discussion

Our pedagogical example in the analysis of the problem is that of the B-1B Lancer bomber. The times used in door movement, launcher rotation, and other constraints are from the performance of the B-1. However, these values are readily changed to match those of other platforms, and are therefore general enough to provide us with a proof of the concept.

3.2.1 Constraints. The constraints in this problem come from the weapons themselves. Each weapon has a set of tolerances which must be met for safety reasons. The tolerances are in terms of air speed, altitude, and bay door position. Each tolerance is further modified by the actual bay location on the aircraft that contains the weapon.

The first constraint is that of air speed. Certain weapons cannot be dropped above a specified speed due to the possiblity of flyback. Flyback is when a weapon's

Weapon Type	Release Bay	Door Config	Restrictions
CBU-103	FWD (10 weapons)	Full	
CBU-103	MID (10 weapons)	Full	FWD Bay doors open full
CBU-103	AFT (5 weapons)	Full	FWD or MID bay doors open full, weapons on C rack only
CBU-103	AFT (10 weapons)	Full	FWD or MID bay doors open full

Table 3.1: Example of Bay Door Constraints

natural lift characteristics combined with the airspeed when it is dropped can allow it fly up into the aircraft, a major safety hazard.

The next weapon specific constraint is that of altitude. While less of a constraint and more a recommendation to achieve full performance, it is still necessary to plan with it as a constraint. This allows the aircraft to enjoy the full benefits of range from the weapon, decreasing threat visibility time.

The most important constraint is that of bay door position. The bay door position varies both by weapon and by bay location. For an example of this constraint, see Table 3.1 from [69].

3.2.2 Weapon Process. The operation of the weapon release is as follows: The weapon is initialized by applying power through the 1760 cable. Targeting data is downloaded from the avionics to the weapon through the weapons interface unit. When valid data is resident in the weapon, it is now live and able to guide itself towards the target to the best of its ability.

In order to drop the weapon, the bay doors on the aircraft must be positioned according to the constraints for the weapon. There is a minimum cycle time which must be allowed for the doors to transition to the desired position prior to weapon release. Additionally, the weapon must be placed in a position on the carrying unit, or rack, where it can be dropped. Problems in this instance include being blocked by other weapons, hung stores, or not being able to rotate into position to be release.

Beginning Position	Ending Position	Time (ms)
Closed	Part Open	5100
Closed	Full Open	7000
Part Open	Closed	5100
Part Open	Full Open	12100
Full Open	Closed	7000
Full Open	Part Open	12100

Table 3.2: Door Movement Times

3.2.3 Door timing.

Since one of the primary motivations for this effort is reducing the amount of time that the bay doors are open, the amount of time that doors spend moving is important. According to [17], the amount of time it takes to move a door into position is given in Table 3.2.

It is also important to note the amount of time that is required for a Multi-Purpose Rotary Launcher (MPRL) to change between stations is 4500ms. When the lancher is already moving, for example from station two to station three in a cycle of station one to four, each additional station after the first adds 4400ms to the rotation time. For the purposes of this project, all times are given in milliseconds. This applies to all of the fitness functions, and allows for times to be given in a single format, with less need to check that the same units are being used across the entire project.

3.2.4 Chromosomes.

The chromosome representation for the problem can be seen in Equation 3.1.

$$C_i = \{\omega_1, ..., \omega_k\} \tag{3.1}$$

Fundamentally, each chromosome C defines an ordered list of weapons (ω), where the order is the release schedule for that weapon. This type of chromosome representation means that we must be careful when applying the mutation operator to the chromosome. In order to prevent infeasible individuals (such as those without certain weapons or duplicates), we implement a static penalty function as a constraint handling technique.

An alternative to this chromosome representation is using a chromosome that consists solely of door positions. This chromosome representation has the advantage of focusing the entire effort on the primary goal of reducing the amount of time a bay door is open. However, by using a chromosome that includes an entire weapon representation as we choose to do, we can look at additional factors in the scheduling such as the weapon position within the bay, individual weapon safety concerns, and timing intervals both between bays and individual weapons.

3.3 Fitness Functions

The proper operation of a genetic algorithm depends on the design and implementation of fitness functions. In this section, we discuss the design of the fitness functions for our problem.

3.3.1 Time Fitness Function. Our primary focus is on reducing the amount of time the bay doors are open. Therefore, the first fitness function we address is that of how long a particular schedule allows the doors to be open.

The formulation for our problem is an algebraic equation. It includes all of the times which must be accounted for in releasing a weapon.

$$\Gamma(S) = \Sigma_{i=1}^{k}(\tau_d(\omega_i) + \tau_r(\omega_i) + \tau_t(\omega_i) + \tau_s(\omega_i)) \tag{3.2}$$

Where τ_d is the time it takes to position the doors correctly from the previous position, τ_r is the time it takes for the weapon to be in position, τ_t is the time needed for data transfer, and τ_s is the minimum safe time which must be allowed for the weapon to release properly.

Given ω_i, a weapon tuple, is defined as

$$\omega = \{t, s, a, d, b, p, r\} \tag{3.3}$$

The weapon tuple parameters are as follows: type (t), airspeed (s), altitude (a), bay door position (d), bay number (b), weapon position (p), and rack type (r).

The fitness is calculated by analyzing the schedule first in terms of dropping the weapons one after another, with only the safety time between them. In this optimal model, the assumptions are made that the bay doors are in the correct position, all weapons have their data, and they are all in the correct position. This value only needs to be calculated once to determine the measure that all individual schedules will be measured against.

A schedule is evaluated using the algebraic model for weapon releases. This is computationally difficult since door position and rack movement must be taken into account. Once a schedule's makespan has been computed, the fitness of the schedule is calculated by subtracting the optimal time from it. The difference is the fitness, which can be clearly seen to be better the lower it is.

Based upon the above discussion, our fitness function for this problem is

$$f_1 = t_s - t_o \tag{3.4}$$

Where t_o is the optimal time (with the assumptions) and t_s is the time for a given schedule. The calculations are

$$\begin{aligned} t_o &= \Sigma_{i=1}^{n} \tau_s(\omega_i) & (3.5) \\ t_s &= \Sigma_{i=1}^{n} (\tau_d(\omega_i) + \tau_r(\omega_i) + \tau_t(\omega_i) + \tau_s(\omega_i)) & (3.6) \end{aligned}$$

Where tau_s is the minimum interval required between weapon releases, tau_d is the time required for door movement, tau_r is the time required for weapon rack movement, and tau_t is the time for data transfer and weapon initialization.

3.3.2 Movement Fitness Function. An important consideration in the wear of the doors is not only the amount of time that they spend in the open position, but also the number of times that the door moves. As a pedagogical example, consider a sequence in which a weapon must be dropped from all three bays. The weapon in the aft bay requires that the mid and forward bay doors be closed, and the weapon in the mid bay requires that the forward bay door be open. Now, a schedule in which the forward weapon was first, followed by the aft weapon, and the mid bay weapon last needs to be evaluated differently from the forward-mid-aft schedule.

The primary focus of this fitness function is the number of door movements which must be made. As this number increases, the overall fitness decreases. We accomplish this be using a simple linear function in which there is a set value per door movement. As more doors need to move, this value increases, representing a decrease in the fitness.

$$f_2 = \Sigma_{i=1}^{n} \omega_i (\Sigma_{j=1}^{3} (d_j)) \tag{3.7}$$

Where d_i is 1 if a door moves and 0 if it does not for each weapon. If all three doors needed to move, the value added would be 3 for that particular weapon.

3.3.3 Safety Fitness Function. The final fitness function we use has to do with the constraints of needing certain altitudes and/or aircraft velocities for certain weapons. If the aricraft's current airspeed is too high, or the current altitude is too low for a weapon in a schedule, the schedule is penalized.

$$f_3 = \Sigma_{i=1}^{n} (p_a(\omega_i)) + \Sigma_{i=1}^{n} (p_v(\omega_i)) \tag{3.8}$$

Where ω_i is a weapon and p is the penalty for being out of constraints. The penalty variable p_a is the altitude penalty, while p_v is the velocity penalty. As more weapons are discovered to be out of the required flight profile, the penalties accumulate. This means that a schedule in which, for example, only the weapons in the aft bay need a lower airspeed will have a higher fitness than a schedule in which the aircraft is traveling too fast to safely drop any weapons.

The value for p in this function is determined by the values of the constraints surrounding the problem. One such value is the amount of time that it takes for a specific weapon to be rotated or otherwise placed into position to be released. These times vary from as low as 0 milliseconds to 9000 milliseconds. Additionally, the amount of time that it takes to move door positions was considered. Therefore, the penalty value for both altitude and velocity was set in the midrange of the various movement times, at 4500ms.

3.4 Summary

In this chapter, we discuss the specifics of the problem domain and outline the constraints inherent to the problem. Additional background material is presented and analyzed. Based upon the discussion, we formulate three fitness functions to be used in evaluating candidate solutions in the genetic algorithm. Using the discussion from this chapter, we next implement specific algorithms and present experimental designs.

IV. Implementation and Experimental Design

The focus of this chapter is the actual implementation of the problem domain design discussed in Chapter III. We cover the fundamental implementation of GENMOP with particular emphasis on problem specific code. Additionally, the experimental goals and design for the completion of the project is discussed in section 4.4. The GenMOP algorithm is introduced in section 2.3.5. Section 4.1 discusses the problem specific development and implementation of GenMOP for our problem.

4.1 GENMOP Implementation

The modularity of GENMOP makes the implementation of our problem specific code require the modification of a small number of files. Building on the efforts of [45], we instantiate a representative chromosome and the necessary operations to build and monitor the chromosome.

The focus in this project is on providing problem specific functionality and operations to the generic operations utilized by GenMOP. Modifications to GenMOP occured in order to reference the data structures provided for this problem. The overall algorithm flow remains the same, with input first being read in from a file. An example of the data format can be seen in Section 4.1.2. With the data read in, the initial population is created and each chromosome is evaluated using the problem-specific fitness function(s). Using the fitness values, the population's members are then ranked using their Pareto-ranking. The most fit individuals in the population are then used in a mating pool, where crossover and mutation are performed. The children created from the mating pool are then evaluated, saved, and merged with previous population (a $\mu+\lambda$ retention scheme). The combined population is reranked according to the Pareto-rankings of the individuals. GenMOP then determines if the required number of generations has been completed, creating another mating pool in the event of further generations being necessary.

4.1.1 GenMOP Program Flow. As discussed in Section 2.3.5, GenMOP is designed to accomodate the solving of multiple objective problems through the use of genetic algorithms. The program flow by which it does this is shown in Figure 4.1.

The basic flow is as follows. The program retrieves the basic information on the number of generations, number of individuals per generation, number of processors, and initial data from data files. An initial population is generated, which is then ranked, normalized to the population size, and saved. The program then asks if it currently on the last generation. If it is the last generation, the program is done. Otherwise, the mating pool is populated from the current population, and crossover is performed. With a new population of children, mutation is performed on them and then they are evaluated. The child and parent populations are then combined, and the total population is ranked before being normalized back to the population size. The new population is then saved, and the program checks if it is the last generation again. The loop continues until the maximum number of generations is reached.

4.1.2 Data Representation. The data in the files used in this program are relatively simple. The first line of the file denotes the number of weapons which are to be scheduled. The next line provides the aircraft's current altitude and speed for the scenario. The remainder of the file is organized in the manner described in Figure 4.2.

Each line in the file after the first two is a weapon. Each weapon is identified by an ID number. The next field identifies which bay the weapon is located in. The maximum airspeed (KCAS) and minimum altitude (in thousands of feet) are then identified, followed by the door configuration for that bay. The interval time (in milliseconds) is given. Finally, the door restrictions for the bays are listed.

One issue with the current data representation is that there is no way to tell from a quick glance at a test file what weapons are due to be scheduled. This is

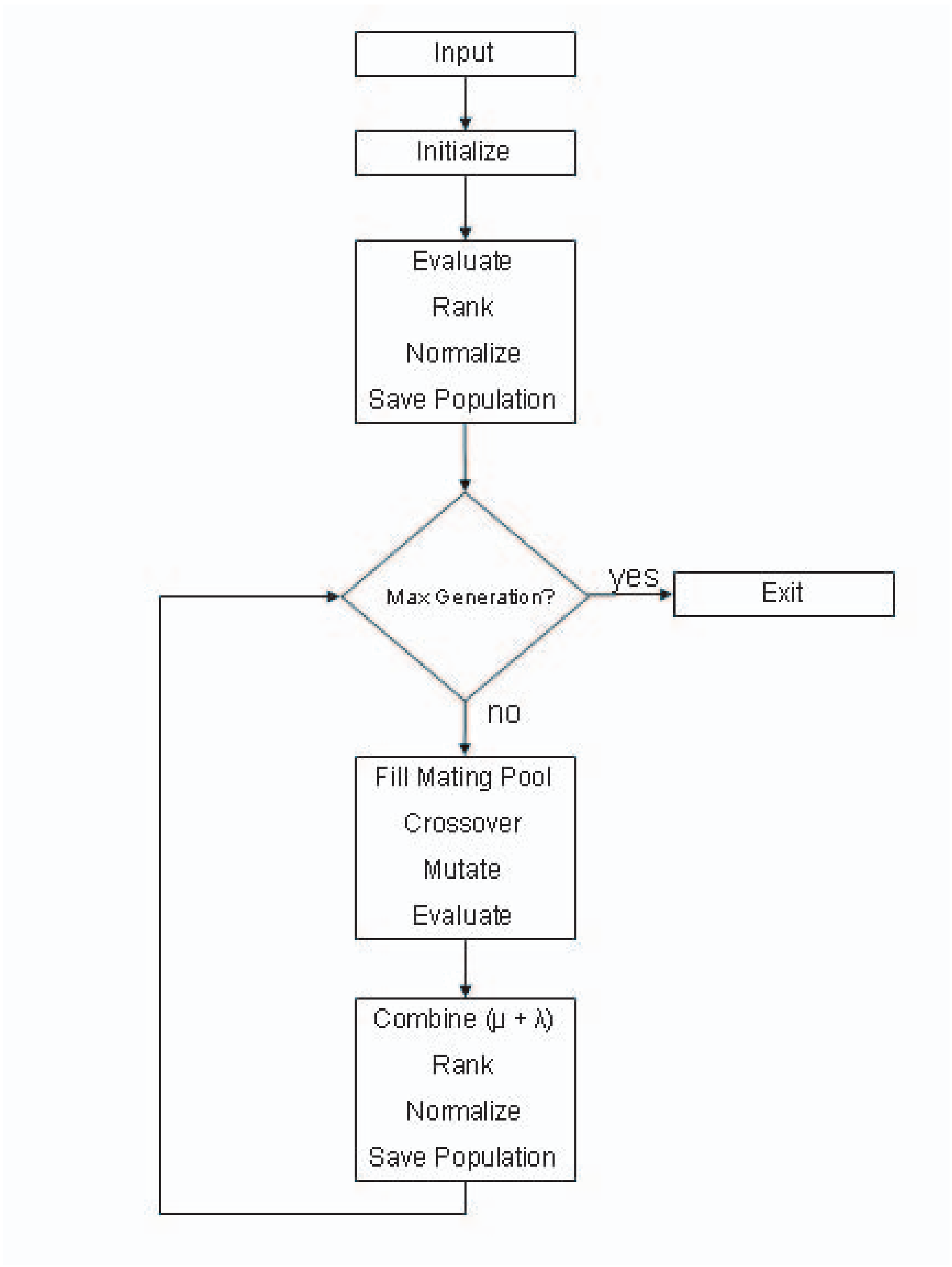

Figure 4.1: GenMOP Program Flow

Weapon ID	Which Bay	Airspeed	Altitude	Door Configuration	Interval	Restriction: FWD	Restriction: MID	Restriction: AFT

Figure 4.2: Weapon Information Layout used for test data sets

due to the design decision to not provide the details of each weapon to the program. Ideally, the solution implementation would have a table of the weapons with the indivdual restrictions. An additional upgrade to the current format would deal with having an aircraft loadout available, with different subsets of the loadout being used as the desired scheduling problem.

4.2 Problem Specific Implementation

In this section, we examine the critical, and specific, code decisions which were made to implement our solution.

4.3 Data Acquisition

This part of our implementation provides the means for using multiple test files and alternate inputs. First, our global data structure contains a pointer to the defined `weapon_data` type. Within the `read_weapon_file` procedure, the pointer is initialized to point to an array of `weapon_data`, the size of which is given by the first line in the data file as described in 4.1.2. After retrieving and setting the aircraft's altitude and speed to be used in fitness function f_3 per the discussion in Section 3.3.3, the procedure then populates the array with the values from the file.

This array is what is referenced in order to create individuals in the population. The array also contains the necessary data for the first and second fitness functions to perform checking upon an individual in a certain generation.

4.3.1 Chromosome Formulation. A chromosome is implemented in the following manner. The first n items in the chromosome are the weapon identification numbers. After the nth weapon, three extra data fields are provided. These fitness function data fields maintain the values from the three fitness functions for evaluation and output. The chromosome can be seen in Equation 4.1.

$$C_i = (\omega_1|\omega_2|\ldots|\omega_n|f_1|f_2|f_3) \tag{4.1}$$

Where C_i is a chromosome, ω denotes a weapon identification number, and f is a fitness function value that is computed by GenMOP.

A chromosome for a specific scheduling scenario is a fixed length. The length for a given chromosome is $n+3$, where n is the number of weapons to be scheduled. The final three pieces of a chromosome is the associated fitness data, and is only used for outputting for data anlysis. Different scenarios of weapon deliveries do not necessarily have differing chromosome lengths, but the implementation of GenMOP allows us to specify the length of the chromosome depending on the number of weapons. This means that overall, we have a variable length chromosome within the overall context of the general scheduling problem, but a fixed length chromosome within a specific schedule.

4.3.2 Fitness Functions. With GenMOP modified to provide the correct input data acquisition and chromosome formulation, we move to implementing the fitness functions. For GenMOP, there is in fact only a single, overall fitness function which is called to evaluate individuals. Within the fitness function, the three fitness functions that are discussed and designed in Chapter III are called. The results from the individual fitness functions are stored in the data at the end of each chromosome. The evaluation is then performed on the results of those three functions.

Time Fitness Function. From the discussion in Section 3.3.1, we have a mathematical model for the fitness function. The algorithmic psuedocode view of the model is shown in Algorithm 2. As implemented in GenMOP, it requires the input of a pointer to the original data array and a pointer to the chromosome currently being evaluated.

Algorithm 2 Time Fitness Function

```
1: for $i = 1$ to number_of_weapons do
2:     optimal_time += weapon_to_drop(i).interval
3: end for
4: for $i = 1$ to number_of_weapons do
5:     current_time += door_move_time(weapon_to_drop(i))
6:     current_time += weapon_to_drop(i).interval
7: end for
8: fitness = current_time - optimal_time
```

An additional function is implemented to provide the `door_move_time` value. This function is based on the door move times given in Table 3.2. It analyzes the needed position and the previous door position and returns the amount of time that it requires to get to the new position.

Movement Fitness Function. The algorithm for the second fitness function is shown in Algorithm 3. The implementation is straitforward, with no additional functions needed. It is important that on the first weapon to be evaluated, we do not set the previous door position since we are assuming that all of the doors are closed and therefore must move into the correct position. `previous_door_position` and `current_door_position` are both initialized to the closed position (0,0,0). Algorithmic psuedocode is shown in Algorithm 3.

Safety Fitness Function. The safety function is easy to implement within GenMOP. Two checks are performed, against the maximum speed and minimum altitude respectively. When a weapon's constraints are violated, the penalty for that schedule is increased. The specific value for each penalty is 3500ms and 2500ms respectively, and is chosen due to the relative values of other aspects of dropping the weapons. Specifically, there are intervals that approach 9000ms while going as low as 210ms. The algorithmic psuedocode can be found in Algorithm 4.

Algorithm 3 Movement Fitness Function

```
for $i = 1$ to number_of_weapons do
    if it is not the first weapon then
        set previous_door_position[j] = current_door_position[j]
    end if
    set current_door_position to the restrictions for the current weapon
    check current_door_position against previous_door_position
    if the current position is not the same as the previous position then
        increment fitness2 by 1
    end if
end for
return fitness2
```

Algorithm 4 Safety Fitness Function

```
fitness3 = 0
for $i = 1$ to number_of_weapons do
    if weapon_max_speed ¡ current_speed then
        fitness3 = fitness3 + 3500
    end if
    if weapon_min_altitude ¿ current_altitude then
        fitness3 = fitness3 + 2500
    end if
end for
return fitness2
```

4.4 Design of Experiments

In order to discern whether the algorithm is functioning correctly, it is necessary to perform experiments. We use three sizes of experiments to validate our model. Each experiment consists of a set of weapons which are needed to be dropped. The algorithm should be able to assign and schedule the weapons for drop within a reasonable time.

The first experimental size is small. In this case, we are using weapon taskings such as one weapon from each bay, or a single bay dropping several weapons. The small case provides an easily verifiable baseline for future experiments. When an assignment calls for multiple weapons, it is a medium sized experiment which provides insight into the operation of the algorithm under realistic conditions. Finally, the

Name	File	Description
JDAM Pure	`small_jdam_pure`	All three bays, all JDAM weapons
JASSM Pure	`small_jassm_pure`	FWD & MID bays, all JASSM weapons
JASSM + JSOW	`small_jassm_jsow`	3 JASSM (AFT), 3 JSOW (FWD)
JDAM + JSOW	`small_jdam_jsow`	3 JDAM (MID), 2 JDAM (AFT), 4 JSOW (FWD)

Table 4.1: Small Experiment Descriptions

large experiment consists of dropping half or more of the total weapon loadout from the aircraft.

In the experiments, we are looking at the time it takes for a solution to be found. This time needs to be consistent for certain sizes of problems, as large problems will in fact receive more time to be performed, within limits. The primary focus in the analysis of the data is to accertain that the algorithm is discovering solutions along the Pareto front. In terms of time and of optimality, the results from the experiments should bear out the hypothesized performance of the algorithm.

4.4.1 Small experiments. The focus of the small size experiments is the correct operation of GENMOP. A small experiment is one of two types of possible problem inputs. The first type is one of fewer than ten weapons of varying types and locations to be scheduled. The other type allows for more than ten weapons, however the aircraft in this case is carrying only a single type of weapon. The summary of experiments is shown in Table 4.1

4.4.2 Intermediate experiments. The intermediate experiment classification is used for experiments in which the aircraft is carrying multiple types of weapons, but the configuration is bay pure. These are shown in Table 4.2

4.4.3 Large experiments. In this experiment, there are multiple types of weapons in the same bay, as well as across the aircraft. Table 4.3 shows the experiments.

Name	File	Description
CBU-105 + JDAM + JSOW	`int_cbu105_jdam_jsow`	4 JDAM (FWD), 5 JSOW (MID), 5 CBU-105 (AFT)
JASSM + JSOW + MK-82	`int_jassm_jsow_mk82`	4 JSOW (FWD), 4 JASSM (MID), 8 MK-82 (AFT)
JASSM + JDAM + JSOW	`int_jassm_jdam_jsow`	JSOW (FWD), JASSM (MID), JDAM (AFT)

Table 4.2: Intermediate Experiment Descriptions

Name	File	Description
All Smart Weapons	`large_jassm_jdam_jsow`	Mix of JASSMs, JDAMs, and JSOWs
Smart and Unguided	`large_cbu105_jdam_jsow_mk82`	Mix of various types of smart and unguided weapons

Table 4.3: Large Experiment Descriptions

4.4.4 Real-world scenarios. The B-1B Lancer program has performed mixed load testing as a part of the Block-E computer upgrade program. In order to provide a better idea of the performance of the algorithm, the operational testing (OT) quick look reports are used to generate additional test files. The primary focus of OT was to validate the performance of equipment and software on the aircraft which was modified by the Block-E effort.

There are three tests performed by the 419th Test Squadron at Edwards Air Force Base, CA, that are of particular interest to the results of this effort. While data is available for the various captive carry (CC) missions which were performed with mixed weapon loads, in these scenarios we do not have data on the release order of weapons and therefore cannot compare the results of our algorithm to the schedule used by the aircraft crew. Therefore, we utilize the three results where the weapons were released. These test were performed from 9 Dec 2003 to 24 Feb 2004 and are summarized in Table 4.4.

4.4.5 GenMOP settings. The settings used for all the the experiments are found in Table 4.5. These setting were chosen based upon the experimental results

Name	File	Description
Flex CC #1	`real_flex_1`	2 JASSM (FWD, 3 JSOW (MID), 2 CBU-105 (AFT)
Flex CC #2	`real_flex_2`	2 JASSM (FWD), 4 JSOW (MID), 3 JDAM (AFT)
Flex CC #2b	`real_flex_2b`	2 JASSM (FWD), 4 JSOW (MID), 3 JDAM (AFT)

Table 4.4: Summary of B-1B Operational Testing Flex missions

Parameter	Value
Maximum number of generations	100
Initial population size	100
Mutation rate	0.02
Save Generations	1
Mating pool size	5
Niche radius	0.2

Table 4.5: GenMOP Settings

in [43, 45, 46], where for the size of our general problem the number of maximum generations and individuals can remain moderate. Most of the parameters have previously been explained, however the Save Generations parameter has not. In GenMOP, this is a flag (0 or 1) which tells the program to save the results at the end of each generation or to throw them away. Saving a generation currently consists for writing the individuals of a generation out to a file for later analysis.

4.5 Hardware Configuration

All experiments were performed on the AFIT Aspen cluster. As of the time of this effort, the current configuration is a Linux Beowulf cluster. Each node within the cluster uses dual 1 GHz Pentium III chips with 1GB of RAM. The operating system is Linux 7.3, creating a homogenous environment. Each test was run on a single node, utilizing both processors and the Message Passing Interface[MPI] capabilities of GenMOP.

4.6 Summary

In this chapter, we discuss the implementation of the design decisions that are outlined in Chapter III. The test plan for the instantiated model is outlined, with particular emphasis on the types of experiments that are run. Real world test scenarios are also laid out, and the background for the scenarios described. We proceed to analyze the results from the testing, and to draw conclusions.

V. Analysis of results

In this chapter, we explore the results of the tests given in Chapter IV. The primary focus is on the examination of the Pareto Front, with some additional analysis provided.

5.1 Collection of Data

GenMOP provides for the automated saving of populations. Each individual is printed to a given output file with the following format. The first item is the Pareto ranking of the individual. This is an integer which relates how many other individuals in the population dominated the particular individual in the population. The entire chromosome of the individual is then saved for future analysis, as well as the fitness data that is appended to each chromosome.

Each experiment is run 30 times in order to provide a measure of statistical validity. The number of runs is determined by the number necessary for the central limit theorem to apply as described in [45, 61], allowing the data to achieve an approximately normal distribution. The results of each run are then combined, retaining only the nondominated individuals for visualization.

5.2 Experimental results and Anlysis

For each experiment, the nondominated individuals are charted based on their relative fitness values. With three fitness functions, the graphs are three dimensional. They show the placement of the nondominated individuals within the solution space. By examining the graphs, we therefore determine the pareto front.

From Figures 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, we can see that the pareto landscape of the solutions is relatively simple. While there are only a few points that are called out in the graph, each point actually represents a multitude of schedules which share the same fitness value in the solution landscape. As a representative,

Scheduled Weapon					
One	Two	Three	Four	Five	Six
1	2	3	4	5	6
2	3	6	4	1	5
1	6	3	2	4	5
2	5	6	4	3	1
5	1	6	3	2	4

Table 5.1: Potential schedules with same fitness value

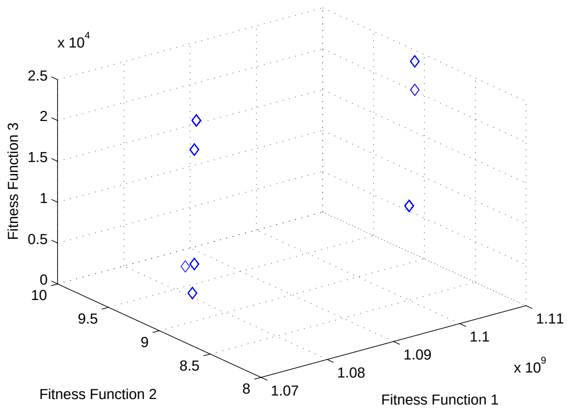

Figure 5.1: Pareto Front graph for `small_jassm_jsow`

we look at the schedules produced for the `small_jassm_jsow` test set shown in Figure 5.1. The total number of schedules that have the same fitness value as the schedule (1,2,3,4,5,6) is 1330. We list some of the representative values in Table 5.1.

5.3 Real world scenario results and analysis

In this section, we look at the computed schedules in regards to the actual drop schedules that were used in B-1 OT.

In the first run, the release sequence was expected to be 2 JSOW/1 JASSM/2 WCMD/1 JASSM. The actual release order was 2 JSOW/2 JASSM/2 WCMD. [60] Both of these schedules are found by the program, with a multitude of other schedules which would have been acceptable. Other schedules which would have been acceptable are found in Table 5.2.

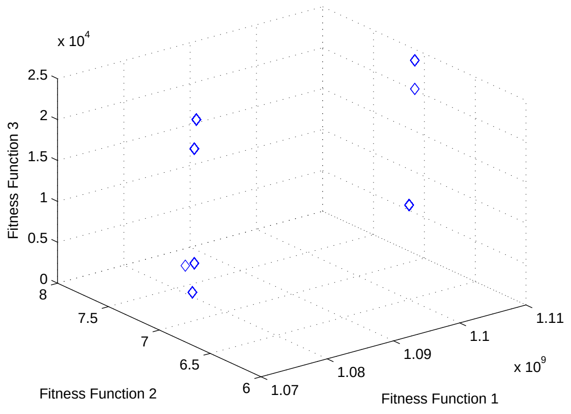

Figure 5.2: Pareto Front graph for `small_jassm_pure`

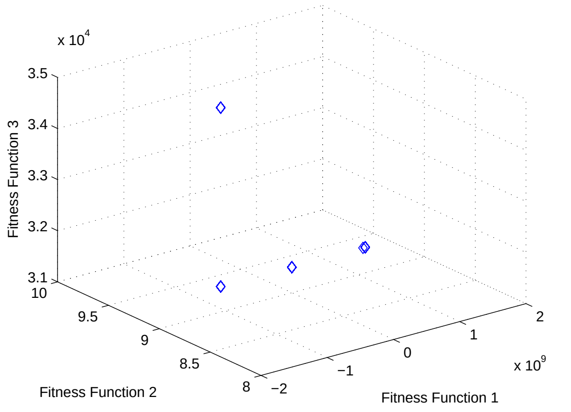

Figure 5.3: Pareto Front graph for `small_jdam_jsow`

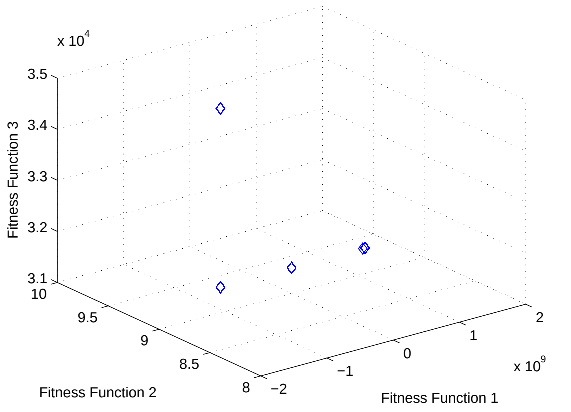

Figure 5.4: Pareto Front graph for `small_jdam_pure`

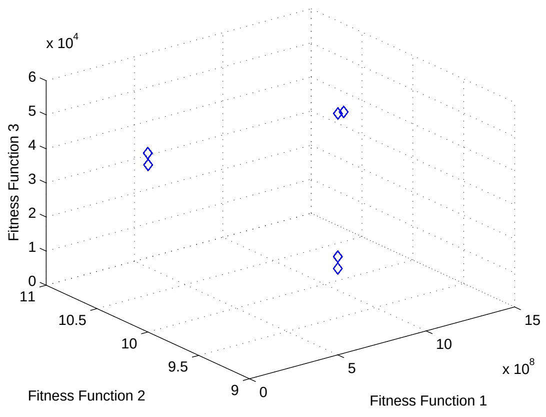

Figure 5.5: Pareto Front graph for `int_cbu105_jdam_jsow`

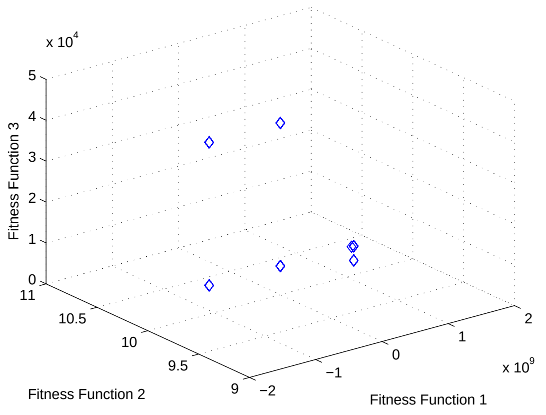

Figure 5.6: Pareto Front graph for `int_jassm_jdam_jsow`

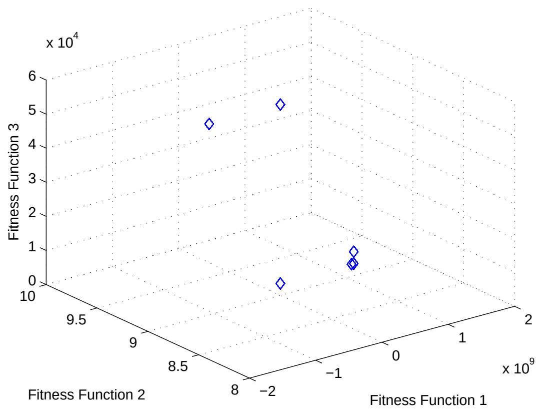

Figure 5.7: Pareto Front graph for `int_jassm_jsow_mk82`

Scheduled Weapon					
One	Two	Three	Four	Five	Six
1	3	2	4	6	5
2	1	3	5	4	6
1	3	4	6	2	5
3	2	6	5	4	1
2	3	4	6	1	5

Table 5.2: Potential schedules for `flexcc_1`

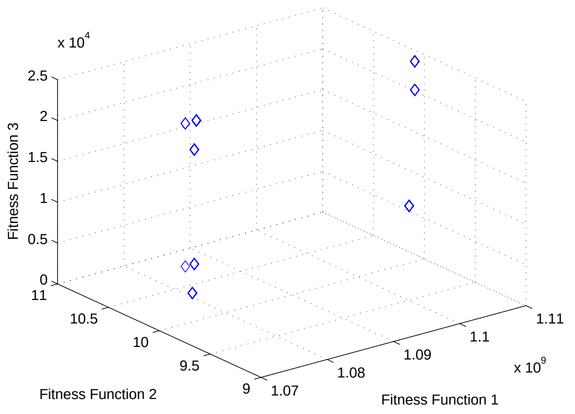

Figure 5.8: Pareto Front graph for `flexcc_1`

It is intersting to note that all of the schedules with the same fitness value as the flown profile begin with weapons from the forward or mid bays. The aft bay weapons are never chosen to begin a viable schedule. From knowledge of the behavior of the aft bay, we know that the most constraints are present on those weapons and for this reason they are not chosen.

This delivery schedule is two JASSMs from the forward bay, four JSOWs from the mid bay, and finally three JDAMs from the aft bay. Again, this schedule is found by our program with a large number of other schedules having the same fitness value. The Pareto front is shown in Figure 5.9.

The delivery profile for this test is similar to the test performed in `flexcc_2`. The same weapon loadout is scheduled to be delivered from a different altitude and mission profile. Specifically, the mission profile calls for a pop-up delivery of

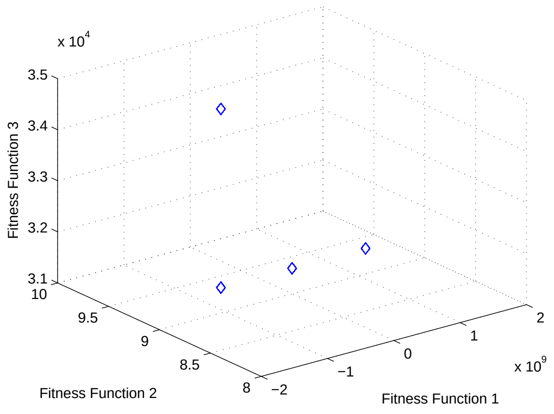

Figure 5.9: Pareto Front graph for `flexcc_2`

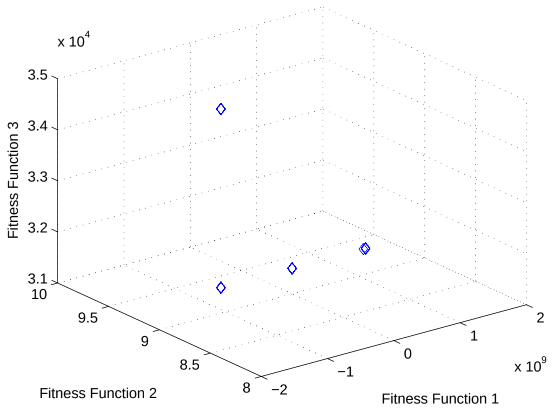

Figure 5.10: Pareto Front graph for `flexcc_2b`

the weapons. If the schedules are evaluated before the proper altitude is reached, the penalty functions will exihibit a disproportionate effect. However, we make the assumption that the aircraft has just achieved the proper altitude, and the results of the experiment are shown in Figure 5.9.

5.4 Analysis

From viewing the raw data and looking at the plots of the pareto fronts, we can see that there are many schedules which have the same fitness value within our solution landscape. This means that while there are some less optimal solutions, a large proportion of the available schedules are as good as any other when it is time to release the weapons.

Therefore, based upon these results, we conclude that the problem is in fact too simple to use an algorithm such as GenMOP on it. This is due to the number of weapons which are to be scheduled at any one time. The primary emphasis was on providing a rescheduling option for Weapon System Operators to use in the event of in-flight rescheduling needs. However, these situations will rarely require more than two or three weapons, and the WSOs are able to provide this capability manually.

5.5 Summary

In this chapter, the method of data collection from GenMOP is described, with particular emphasis on the data required for our problem. The collected data is graphed to determine the Pareto front in each experiment, and analyzed in terms of the scheduling options which the program provides. Finally, we perform an overall analysis on all of the available data, and conclude that the problem is simpler than initial analysis would indicate for our given approximate problem domain model.

VI. Conclusions

This chapter summarizes the work performed in this effort. A review of the original goals is presented with relevant conclusions, building on the results presented in Chapter V. Areas of future work are outlined. Finally, conclusions based on this work are presented.

6.1 Goals

As presented in Chapter I, our objectives are as follows. We will examine them in this section and analyze the project in terms of their accomplishment.

1. Develop a model that encapsulates the problem domain.
2. Develop an effective algorithm which solves the scheduling problem.
3. Evaluate the algorithm for efficiency.
4. Provide the algorithm in portable form for use in multiple environments.

6.1.1 Objective 1: Model Development. The focus of this objective is to communicate the problem domain, and quantify it symbolically, using mathematics. The analysis of the problem domain provides the necessary information to create the mathematical formulation. This goal is accomplished in Chapters II and III. The overall model is mapped to the Job Shop scheduling problem, and various methods of solving the scheduling problem are presented.

6.1.2 Objective 2: Algorithm Development. This objective focuses on developing algorithms capable of providing solutions to the model laid out in objective one. It uses the background information and discussion present in objective one to provide these algorithms. The majority of work on this objective is accomplished in Chapter III. We use an existing algorithm to provide the framework for our work, providing detailed implementation and parameter choices that are based on the problem to be solved.

6.1.3 Objective 3: Algorithm Optimization. Initial testing with the implemented algorithms provides feedback which is used to tune some of the parameters of the program. The primary parameter that is modified is the value in fitness function three. This value controls the penalty that applies to schedules that violate the aircraft and weapon safety constraints of airspeed and altitude. The other parameters that are modified based on initial testing are the number of generations and initial starting individuals in the program.

6.1.4 Objective 4: Algorithm Portability. The conclusions that we draw from the results that we present in Chapter V makes this objective moot. The reason for this objective was to provide something which would be used by the end user in the performance of normal duties. With what is known to be a problem of this simplicity, the complex MOEA algorithm simply is not needed for use in the field. Additionally, the problem domain model as currently designed and implemented is highly static. The actual operation is highly fluid and dynamic, which requires a revisiting of the problem domain to accurately gauge the effect of a dynamic environment on weapon scheduling. Already established practices accomplish the necessary tasks such that this program is not required.

6.2 Future work

In order to provide a useful program, the current implementation of the program must be extended. The model of the operation of the aircraft must be better quantified, and implemented within the code. Currently, a large portion of the model is dependant on factors which are brough in through the data files to be analyzed.

6.3 Summary

In conclusion, this effort demonstrated that while the weapon delivery scheduling problem is complex, it is by no means complex enough to warrant the use of evolutionary computation to solve it. The current methods for solving the problem are entirely satisfactory, and do not require augmentation.

Bibliography

1. Abali, M. Banikazemiand B., L. Herger, and D. K. Panda. “Design Alternatives for Virtual Interface Architecture (VIA) and an Implementation on IBM Netfinity NT Cluster”. *Journal of Parallel and Distributed Computing, Special Issue on Clusters*, in press. URL `ftp://ftp.cis.ohio-state.edu/pub/communication/papers/jpdc_VIA.pdf`.

2. AFRL/MN. “Mission Statement”. URL `http://www.mn.afrl.af.mil/`.

3. et. al., N. Boden. “Myrinet - A Gigabit-per-Second Local-Area Network”. *IEE Micro*, Feb 1995. URL `http://www.myri.com`.

4. ASC/ENOS. *Department of Defense Interface Standard for Aircraft/Store Electrical Interconnection System*, August 2003.

5. Aspen Systems, Inc. “The History Behind Beowulf Clusters”. URL `http://www.aspsys.com/clusters/beowulf/history/`.

6. Bäck, Thomas, David B. Fogel, and Zbigniew Michalewicz (editors). *Evolutionary Computation 1: Basic Algorithms and Operators.* Institute of Physics Publishing, Philadelphia, 2000.

7. Bäck, Thomas, David B. Fogel, and Zbigniew Michalewicz (editors). *Evolutionary Computation 2: Advanced Algorithms and Operators.* Institute of Physics Publishing, Philadelphia, 2000.

8. Bäck, Thomas, David B Fogel, Darrell Whitley, and Peter J Angeline. *Mutation operators*, chapter 32, 237–255. Institute of Physics Publishing, 2000.

9. Balas, E. and A. Vazacopoulos. *Guided local search with shifting bottleneck for job shop scheduling.* Tech Rep. Management Science Research Report MSRR-609, Carnegie Mellon University, 1994.

10. Banikaze, M., B. Abali, and D. K. Panda. “Comparison and Evaluation of Design Choices for Implementing the Virtual Interface Architecture (VIA)”. *Fourth Int’l Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing (CANPC ’00)*, 145–161. 2000.

11. Banzhaf, Wolfgang, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction.* Morgan Kaufmann Publishers, Inc, 1998.

12. Beasley, David. *Possible applications of evolutionary computation*, chapter 2, 4–19. Institute of Physics Publishing, 2000.

13. Becker, Don, Robert Brown, Greg Lindahl, Forrest Hoffman, Putchong Uthayopas, and Kragen Sitaker. “Frequently Asked Questions”. URL `http://www.beowulf.org/overview/faq.html`.

14. Bierwirth, Christian and Dirk C. Mattfeld. “Production Scheduling and Rescheduling with Genetic Algorithms”. *Evolutionary Computation*, 7(1):1–17, 1999. URL `citeseer.ist.psu.edu/bierwirth99production.html`.

15. Bock, Stefan, Hergen Busch, and Otto Rosenberg. “Two New Parallel Algorithms for Solving the Job Shop Problem”. URL `citeseer.ist.psu.edu/367284.html`.

16. Bock, Stefan and Otto Rosenberg. “New Parallel Algorithms for Solving the Job Shop Problem”. URL `citeseer.ist.psu.edu/bock00new.html`.

17. Bolek, Dan. “Door Movement Times”.

18. Booker, Lashon B, David B Fogel, Darrell Whitley, Peter J Angeline, and A E Eiben. *Recombination*, chapter 33, 256–307. Institute of Physics Publishing, 2000.

19. Branke, Jurgen, Thomas Kauler, Christian Schmidt, and H. Schmeck. “A Multi-Population Approach to Dynamic Optimization Problems”. *Adaptive Computing in Design and Manufacture (ACDM 2000)*. 2000. URL `citeseer.ist.psu.edu/658304.html`.

20. Brooks, Michael. “B-1B Quicklook Report: Flex CC #2”, 2003.

21. Brooks, Michael. “B-1B Quicklook Report: Flex CC #2b”, 2003.

22. Buyya, Rajkumar and Mark Baker. *Cluster Computing at a Glance*, volume 1, chapter 1, 3–47. Prentice Hall PTR, 1999.

23. de Castro, Leandro N. and Jonathan Timmis. *Artificial Immune Systems: A New COmputational Intelligence Approach*, chapter 7.4. Springer, 2002.

24. Chandran, R. Vijaya. “An Introduction to Genetic Algorithms”, Nov 1999. URL `http://www.ittc.ku.edu/ rvc/projects/genetic/index.htm`. Bell Labs Development Center, Lucent Technologies.

25. Coello, C. and A. Carlos. “A survey of constraint handling techniques used with evolutionary algorithms”, 1999. URL `citeseer.ist.psu.edu/coello99survey.html`.

26. Coffman Jr., E. G. (editor). *Computer and Job-Shop Scheduling Theory.* John Wiley & Sons, 1976.

27. Coffman Jr., E. G. *Introduction to Deterministic Scheduling Theory*, chapter 1, 1–50. John Wiley & Sons, 1976.

28. Corne, David, Marco Dorigo, and Fred Glover. "Introduction". David Corne, Marco Dorigo, and Fred Glover (editor), *New Ideas in Optimization*, chapter 1, 1–8. McGraw Hill, 1999.

29. Deb, Kalyanmoy. "Genetic Algorithm in Search and Optimization: The Technique and Applications". URL `citeseer.ist.psu.edu/138345.html`.

30. Duvivier, D., P. Preux, and E. Talbi. "Parallel genetic algorithms for optimization and application to NP-complete problem solving". *Int. Workshop on Combinatorics and Computer Science, Brest, France, 1995*. 1995. URL `citeseer.ist.psu.edu/article/duvivier96parallel.html`.

31. Fang, Hsiao-Lan, Peter Ross, and Dave Corne. "A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Re-Scheduling, and Open-Shop Scheduling Problems". Stephanie Forrest (editor), *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, 375–382. Morgan Kaufmann, San Mateo, CA, 1993. URL `citeseer.ist.psu.edu/fang93promising.html`.

32. Fogel, David. *Introduction to evolutionary computation*, chapter 1, 1–3. Institute of Physics Publishing, 2000.

33. Fogelman, Ronald R. *1997 Air Force Issues Book*. United States Air Force, 1997. URL `http://www.af.mil/lib/afissues/1997/`.

34. Fonseca, Carlos M. and Peter J. Fleming. "An Overview of Evolutionary Algorithms in Multiobjective Optimization". *Evolutionary Computation*, 3(1):1–16, 1995. URL `citeseer.ist.psu.edu/article/fonseca95overview.html`.

35. Fonseca, Carlos M. and Peter J. Fleming. "Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms—Part I: A Unified Formulation". *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 28(1):26–37, 1998. URL `citeseer.ist.psu.edu/fonseca98multiobjective.html`.

36. Garrett, C. A. *Optimization of In Situ Aerobic Cometabolic Biremediation of Trichloroehylene-Contaminated Groundwater Using a Parallel Genetic Algorithm*. Master's thesis, Air Force Institute of Technology, 1999.

37. Gonalves, Jos F., Jorge Jos M. Mendes, and Mauricio G. C. Resende. *A hybrid genetic algorithm for the job shop scheduling problem*. Technical Report TD-5EAL6J, AT&T Labs Research, 180 Park Avenune, Florham Park, NJ 07932 USA, Sep 2002. URL `http://www.optimization-online.org/DB_FILE/2002/09/538.pdf`.

38. Hart, Emma and Peter Ross. "The Evolution and Analysis of a Potential Antibody Library for Use in Job-Shop Scheduling". David Corne, Marco Dorigo, and Fred Glover (editor), *New Ideas in Optimization*, chapter 12, 185–202. McGraw Hill, 1999.

39. Holland, John. *Adaptation in Natural and Artificial Systems.* Technical report, MIT Press, 1975.

40. Inc., Voltaire. *InfiniBand: The Next Step in High Performance Computing.* Technical report, Voltaire Inc., 2003. URL `www.hitachi-hitec.com/jyouhou/network/@gif/voltaire_20hpc.pdf`.

41. Jain, A. and S. Meeran. *A state-of-the-art review of job-shop scheduling techniques.* Technical report, University of Dundee, 1998. URL `citeseer.ist.psu.edu/jain98stateart.html`.

42. Karp, Gerald. *Cell and Molecular Biology: Concepts and Experiments.* John Wiley & Sons, Inc., second edition, 1999.

43. Keller, Traci A. *Optimization of a Quantum Xascade Laser Operating in the Terahertz Frequency Range Using a MultiObjective Evolutionary Algorithm.* Master's thesis, Air Force Institute of TEchnology, 2004.

44. Kennedy, Harold. "Air Force Seeks to Upgrade Close Air Support Fleet". *National Defense*, Jul 2004.

45. Kleeman, Mark and Gary Lamont. "Solving the Sircraft Engine Maintenance Scheduling Problem Using a Multi-objective Evolutionary Algorithm", 2005.

46. Knarr, Mark R. *Optimizing of In Situ Bioremediation Technology to Manage Perchlorate- Contaminated Groundwater.* Master's thesis, Air Force Institute of Technology, Mar 2003. URL `http://handle.dtic.mil/100.2/ADA415320`.

47. Kwok, Yu-Kwong and Ishfaq Ahmad. *Parallel Program Scheduling Techniques*, volume 1, chapter 23, 553–578. Prentice Hall PTR, 1999.

48. Kwok, Yu-Kwong and Ishfaq Ahmad. "Static scheduling algorithms for allocating directed task graphs to multiprocessors". *ACM Computing Surveys*, 31(4):406–471, 1999. URL `citeseer.ist.psu.edu/kwok99static.html`.

49. Lyons, Francis. "790 Final Report", 2004. Final report for AFIT class CSCE790.

50. Lyons, Francis. "886 Final Report", 2004. Final report for AFIT class CSCE886.

51. Martin, Paul and David B. Shmoys. "A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem". W. H. Cunningham, S. T. McCormick, and M. Queyranne (editors), *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization, IPCO'96*, 389–403. Vancouver, British Columbia, Canada, 1996. URL `citeseer.ist.psu.edu/martin96new.html`.

52. Merkey, Phil. "Beowulf History". URL `http://www.beowulf.org/overview/history.html`.

53. Meuer, Hans, Erich Strohmaier, Horst Simon, and Jack Dongarra. "23rd Edition of TOP500 List of World's Fastest Supercomputers Released: Japan's

Earth Simulator Enters Third Year in Top Position". Internet, Jun 2004. URL `http://www.top500.org/news/articles/article_21.php`.

54. Michalewicz, Zbigniew. "A survey of constraint handling techniques in evolutionary computation methods". John R. McDonnell, Robert G. Reynolds, and David B. Fogel (editors), *Proc. of the 4th Annual Conf. on Evolutionary Programming*, 135–155. MIT Press, Cambridge, MA, 1995. URL `citeseer.ist.psu.edu/michalewicz95survey.html`.

55. Michalewicz, Zbigniew. *Decoders*, chapter 8, 49–55. Institute of Physics Publishing, 2000.

56. Michalewicz, Zbigniew. *Introduction to constraint-handling techniques*, chapter 6, 38–40. Institute of Physics Publishing, 2000.

57. Michalewicz, Zbigniew. *Other constraint-handling methods*, chapter 10, 69–73. Institute of Physics Publishing, 2000.

58. Michalewicz, Zbigniew. *Repair algorithms*, chapter 9, 56–68. Institute of Physics Publishing, 2000.

59. Michaud, Steven. *Solving the Protein Structure Prediction Problem With Fast Messy Genetic Algorithms (Scaling the Fast Messy Genetic Algorithm to Medium-Sized Peptides by Detecting Secondary Structures)*. Master's thesis, Air Force Institute of Technology, 2001.

60. Middleton, Angela. "B-1B Quicklook Report: Flex CC #1", 2003.

61. Milton, J Susan and Jesse C Arnold. *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences*. McGraw-Hill College, 4th edition, 2002. ISBN 007246836X.

62. Nowicki, E. and C. Smutnicki. "A fast taboo search algorithm for the job shop problem". *Management Scienc*, 42(6):797–813, 1996.

63. Rudolph, Günter. *Evolution strategies*, chapter 9, 81–88. Institute of Physics Publishing, 2000.

64. Sethi, Ravi. *Algorithms for Minimal-Length Schedules*, chapter 2, 51–99. John Wiley & Sons, 1976.

65. Shmoys, Stein, and Wein. "Improved Approximation Algorithms for Shop Scheduling Problems". *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*. 1991. URL `citeseer.ist.psu.edu/shmoys94improved.html`.

66. Smith, Alice and David Coit. *Penalty functions*, chapter 6, 41–49. Institute of Physics Publishing, 2000.

67. Snoek, Marko. "Anticipation Optimization in Dynamic Job Shops". Jürgen Branke and Thomas Bäck (editors), *Evolutionary Algorithms for Dynamic Optimization Problems*, 43–46. San Francisco, California, USA, 7 July 2001. URL `citeseer.ist.psu.edu/snoek01anticipation.html`.

68. Spillman, Mark. "B-1B achieves triple munitions drop, target kill". *Air Force News*, May 2002. URL `http://www.af.mil/news/May2002/n20020509_0749.asp`.

69. SPO, ASC/YD B-1B. "Door Restrictions". Microsoft Excel, 2004.

70. Ullman, J. D. *Complexity of Sequencing Problems*, chapter 4, 139–164. John Wiley & Sons, 1976.

71. Vaessens, R., E. Aarts, and J. Lenstra. "Job-Shop Scheduling by Local Search", 1994. URL `citeseer.ist.psu.edu/vaessens94job.html`.

72. Vázquez, Manuel and L. Darrell Whitley. "A Comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem", 2000. URL `http://www.cs.colostate.edu/ genitor/Pubs.html`.

73. Vázquez, Manuel and L. Darrell Whitley. "A Comparison of Genetic Algorithms for the Static Job Shop Scheduling Problem", 2000. URL `citeseer.ist.psu.edu/528189.html`.

74. Viega, J. "Practical Random Number Generation in Software". *Proc. 19th Annual Computer Security Applications Conference.* Dec 2003. URL `http://www.acsac.org/2003/papers/79.pdf`.

75. Watson, J., J. Beck, A. Howe, and L. Whitley. "Toward an understanding of local search cost in job-shop scheduling". *Proceedings of the Sixth European Conference on Planning (ECP'01), 2001.* 2001. URL `citeseer.ist.psu.edu/article/watson01toward.html`.

76. Watson, Jean-Paul and J. Christopher Beck. "Toward a Descriptive Model Of Local Search Cost in Job-Shop Scheduling". URL `citeseer.ist.psu.edu/454338.html`.

77. Wu, Annie S., Han Yu, Shiyuan Jin, Kuo-Chi Lin, and Guy Schiavone. "An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling". *IEEE Transactions on Parallel and Distributed Systems*, 15(9):824–834, Sep 2004.

Index

The index is conceptual and does not designate every occurrence of a keyword. Page numbers in bold represent concept definition or introduction.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)* 21–03–2005	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sept 2003 — Mar 2005

4. TITLE AND SUBTITLE

Weapon Release Scheduling from Multiple-Bay Aircraft using Multi-Objective Evolutionary Algorithms

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Francis R. Lyons, 1Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management
2950 Hobson Way, Bldg 641
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/05-04

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Lloyd Reshard
850-882-8876 x3209
AFRL/MNAV
101-348 Eglin Blvd
Eglin Air Force Base, FL 32542

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The United States Air Force has put an increased emphasis on the timely delivery of precision weapons. Part of this effort has been to use multiple bay aircraft such the B-1B Lancer and B-52 Stratofortress to provide Close Air Support and responsive strikes using 1760 weapons. In order to provide greater flexibility, the aircraft carry heterogeneous payloads which can require deconfliction in order to drop multiple different types of weapons. Current methods of deconfliction and weapon selection are highly crew dependant and work intensive.
This research effort investigates the optimization of an algorithm for weapon release which allows the aircraft to perform deconfliction automatically. This reduces crew load and response time in order to deal with time-sensitive targets. The overall problem maps to the Job-Shop Scheduling problem. Optimization of the algorithm is done through the General Multiobjective Parallel Genetic Algorithm (GENMOP). We examine the results from pedagogical experiments and real-world test scenarios in the light of improving decision making. The results are encouraging in that the program proves capable of finding acceptable release schedules, however the solution space is such that applying the program to real world situations is unnecessary. We present visualizations of the schedules which demonstrate these conclusions.

15. SUBJECT TERMS

Weapon release scheduling, genetic algorithms, job-shop scheduling

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Gary B. Lamont
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U	UU	65	(937) 255–3636, ext 4718

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18